

This paper was accepted for publication in Circuits, Systems & Signal Processing journal. A copyright may be transferred without notice.

Generation and Analysis of Constrained Random Sampling Patterns

Jacek Pierzchlewski, Thomas Arildsen

Signal and Information Processing,
Department of Electronic Systems, Aalborg University,
Fredrik Bajers Vej 7, DK-9220 Aalborg, Denmark
jap@es.aau.dk, tha@es.aau.dk

October 8, 2015

Abstract

Random sampling is a technique for signal acquisition which is gaining popularity in practical signal processing systems. Nowadays, event-driven analog-to-digital converters make random sampling feasible in practical applications. A process of random sampling is defined by a sampling pattern, which indicates signal sampling points in time. Practical random sampling patterns are constrained by ADC characteristics and application requirements. In this paper we introduce statistical methods which evaluate random sampling pattern generators with emphasis on practical applications. Furthermore, we propose a new random pattern generator which copes with strict practical limitations imposed on patterns, with possibly minimal loss in randomness of sampling. The proposed generator is compared with existing sampling pattern generators using the introduced statistical methods. It is shown that the proposed algorithm generates random sampling patterns dedicated for event-driven-ADCs better than existed sampling pattern generators. Finally, implementation issues of random sampling patterns are discussed.

keywords: Analog-digital conversion, Compressed sensing, Digital circuits, Random sequences, Signal sampling

1 Introduction

In many of today's signal processing systems there is a need for random signal sampling. The idea of random signal sampling dates back to early years of the study on signal processing [1]. Signal reconstruction methods for this kind of sampling were studied [2], there are practical implementations of signal acquisition systems which employ random nonuniform sampling [3, 4, 5]. Recently, this method of sampling has received more attention hence to a relatively new field

of signal acquisition known as compressed sensing [6, 7]. It was shown that in many compressed sensing applications the random sampling is a correct choice for signal acquisition [8]. The random sampling gives a possibility to sample below Nyquist rate, which lowers the power dissipation and reduces the number of samples to be processed. A process of random sampling is defined by a sampling pattern, which indicates signal sampling points in time. Generation and analysis of random sampling patterns which are dedicated to be implemented in analog-to-digital converters is a subject of this work.

In practice, sampling according to a given sampling pattern is realized with analog-to-digital converters [9, 10]. Currently, there are available event-driven analog-to-digital converters, which are able to realize random sampling [4, 11]. These converters have certain practical constraints coming from implementation issues, which consequently puts implementation-related constraints on sampling patterns. These constraints concern minimum and maximum time intervals between adjacent sampling points, e.g. Wakin et. al. in their work [5] used a random nonuniform sampling pattern with minimum and maximum intervals between adjacent sampling points. Furthermore, there are application-related constraints which concern stable average sampling frequency of sampling patterns, equal probability of occurrence of possible sampling points, and uniqueness of generated patterns.

The problem which this work solves is composed of two parts. Firstly, how to evaluate different sampling pattern generators with emphasis on practical applications? The early work on estimation of random nonuniform sampling patterns was done by Marvasti [12]. Wakin et. al. [5] looked for a sampling pattern with the best (most equal) histogram of inter-sample spacing. Gilbert et. al [13] proposed to choose a random sampling pattern based on permutations. To the best of our knowledge, there is no scientific work published which concerns multiparameter statistical analysis of random sampling patterns. Due to the constantly increasing available computational power it has become possible to analyze random pattern generators statistically within a reasonable time frame. Statistical parameters which assess random sampling pattern generators with respect to the constraints described above are described in this paper.

The second problem discussed in this paper is how to construct a random sampling pattern generator which generates patterns with a given number of sampling points, and given intervals between sampling points, with possibly minimum loss in randomness? The well known random sampling pattern generators are Additive Random Sampling (ARS) and Jittered Sampling (JS) [14, 15]. However, these sampling pattern generators do not take into account the mentioned implementation constraints, which is an obstacle in practical applications. There have been some attempts to generate more practical sampling patterns. Lin and Vaidyanathan [16] discussed periodically nonuniform sampling patterns which are generated by employing two uniform patterns. Bilinskis et al. in [17] introduced a concept of correlated additive random sampling, which is a modification of the ARS. Papenfuß et al. in [18] proposed another modification of the ARS process, which was supposed to optimally utilize the ADC. Ben-Romdhane et al. [19] discussed a hardware implementation of a nonuniform pseudorandom

clock generator. Unfortunately, none of the proposed sampling pattern generators are designed to address all the implementation constraints. This paper proposes a sampling pattern generator which is able to produce constrained random sampling patterns dedicated for use in practical acquisition systems. The generator is compared with existing solutions using the proposed statistical parameters. Implementation issues of this generator are discussed.

The paper is organized as follows. The problem of random sampling patterns generation is identified in Section 2. Statistical parameters for random pattern generators are proposed in Section 3. A new random sampling pattern generator for patterns to be used in practical applications is proposed in Section 4. The proposed generator is compared with existing generators in Section 5. Some of the implementation issues of random sampling patterns are discussed in Section 6. Conclusions close the paper in Section 7. The paper follows the reproducible research paradigm [20], therefore all of the code associated with the paper is available online [21].

2 Problem formulation

2.1 Random sampling patterns

This paper focuses on generation and analysis of random sampling patterns. The purpose of this Section is to formally define a sampling pattern and its parameters, and to discuss requirements for sampling patterns and sampling pattern generators. A sampling pattern \mathbb{T} is an ordered set (sequence) with K_s fixed sampling time points:

$$\mathbb{T} = \{t_1, t_2, \dots, t_{K_s}\} \quad (1)$$

where the sampling time points t_k are real numbers ($t_k \in \mathbb{R}$, $k = \{1, 2, \dots, K_s\}$). Elements of such a set \mathbb{T} must increase monotonically:

$$t_1 < t_2 < \dots < t_{K_s} \quad (2)$$

Time length τ of a sampling pattern is equal to the time length of a signal or a signal segment on which the sampling pattern is applied. The time length τ may be higher than the last time point in a pattern: $\tau \geq t_{K_s}$.

Any sampling point $t_k \in \mathbb{T}$ is a multiple of a sampling grid period T_g :

$$t_k = kT_g, \quad k \in \mathbb{N}^* \quad (3)$$

where \mathbb{N}^* is the set of natural numbers without zero. The sampling grid is a set:

$$\mathbb{G} = \{T_g, 2T_g, \dots, K_g T_g\}, \quad K_g = \left\lfloor \frac{\tau}{T_g} \right\rfloor \quad (4)$$

where K_g is the number of sampling grid points in a sampling pattern, and $\lfloor \cdot \rfloor$ signifies the floor function, which returns the largest integer lower or equal to the function argument. It can be stated that a pattern \mathbb{T} is a subset of a

grid set \mathbb{G} ($\mathbb{T} \subset \mathbb{G}$). The sampling grid period T_g describes the resolution of the sampling process. In practice, the lowest possible sampling grid depends on the performance of the used ADC, its control circuitry, and the clock jitter conditions [4, 9, 10]. A sampling pattern may be represented as indices of sampling grid:

$$\mathbb{T}' = \{t'_1, t'_2, \dots, t'_{K_s}\}, \quad t'_k = \frac{t_k}{K_g} \quad (5)$$

Let us define a set \mathbb{D} which contains $K_s - 1$ intervals between the sampling points:

$$\mathbb{D} = \{d_1, d_2, \dots, d_{K_s-1}\}, \quad d_k = t_{k+1} - t_k \quad (6)$$

If all the intervals are equal ($\forall k : d_k = T_s$), then \mathbb{T} is a uniform sampling pattern with a sampling period equal to T_s . If the time intervals are chosen randomly, then \mathbb{T} is a random sampling pattern.

A random sampling pattern \mathbb{T} is applied to a signal $s(t)$ of length τ :

$$\mathbf{y}[k] = s(t_k), \quad t_k \in \mathbb{T} \quad (7)$$

where $\mathbf{y} \in \mathbb{R}^{K_s}$ is a vector of observed signal samples. The average sampling frequency f_s of a random sampling pattern depends on the number of sampling time points in the pattern:

$$f_s = \frac{K_s}{\tau} \quad (8)$$

An example of a random sampling pattern is shown in Fig. 1.

2.2 Random patterns generation problem

Let us denote a nontrivial problem $\mathcal{P}(N, \tau, T_g, f_s^\dagger, t_{\min}, t_{\max})$ of generation of a multiset (bag) \mathbb{A} with N random sampling patterns. The time length of sampling patterns is τ , grid period is T_g . The requested average sampling frequency of patterns is f_s^\dagger , minimum and maximum intervals between sampling points are t_{\min} and t_{\max} respectively. The problem \mathcal{P} is solved by random sampling pattern generators. The generators should meet requirements given in 2.4, and all the produced sampling patterns must meet the requirements given below in 2.3.

2.3 Requirements for random sampling patterns

2.3.1 Frequency stability

A random sampling pattern generator must produce sampling patterns with a requested average sampling frequency f_s^\dagger . If the average sampling frequency f_s is lower than the requested sampling frequency, then the quality of signal reconstruction may be compromised. On the contrary, higher sampling frequency f_s than the requested f_s^\dagger causes unnecessary power consumption.

2.3.2 Minimum and maximum time intervals

A requirement for minimum interval t_{\min} between sampling points comes from the ADC technological constraints [9, 10, 4, 11]. Violation of this requirement may render the sampling pattern impossible to implement with a given ADC. Similarly, there may be a requirement of maximum interval between samples t_{\max} . Generating an adequate random sampling pattern is realizable if $t_{\min} \leq T_s^\dagger$ and $t_{\max} \geq T_s^\dagger$, where $T_s^\dagger = 1/f_s^\dagger$ is the requested average sampling period.

2.3.3 Unique sampling points

As stated in (2), sampling points in a given sampling pattern \mathbb{T} cannot be repeated. Repeated sampling points do not make practical sense since a signal can be sampled only once in a given time moment. If a sampling pattern contains repeated sampling points, then a dedicated routine must remove these repeated points.

2.4 Requirements for random sampling pattern generators

2.4.1 Uniform probability density function for grid points

As described in 2.1, a sampling pattern \mathbb{T} is an ordered set which is a subset of a grid \mathbb{G} . In other words, sampling points are drawn from a pool of grid points. The sampling pattern generator should not favor any of the sampling grid points. Ideally, all of the sampling points should be equi-probable.

2.4.2 Pattern uniqueness

Repeated sampling patterns generate unnecessary processing overhead, especially if sampling patterns are generated offline and further processed (Fig. 5). An additional search routine which removes replicas of sampling patterns must be implemented in this case. Therefore, the ideal random sampling pattern generator should not repeat sampling patterns unless all the possible sampling patterns have been generated.

3 Statistical evaluation of random sampling pattern generators

In this Section we propose statistical parameters for evaluation of a tested random sampling pattern generator. Aim of these parameters is to assess how well sampling patterns produced by the evaluated generator cope with the requirements described in 2.3 and 2.4. These parameters are to be computed for a bag \mathbb{A} of N patterns produced by the evaluated generator, the parameters are computed using the Monte Carlo method. It is checked if every generated sampling pattern fulfills requirements given in 2.3 and if a generated bag (multiset) of

sampling patterns fulfill requirements given in the 2.4. According to our best knowledge, similar statistical evaluation has never been introduced before.

3.1 Frequency stability error parameters

Let us introduce a statistical parameter indicating how well the evaluated generator fulfills the imposed requirement of the requested average sampling frequency f_s^\dagger (2.1):

$$e_f = \frac{1}{N} \sum_{n=1}^N \left(\frac{f_s^\dagger - f_s^{(n)}}{f_s^\dagger} \right)^2 = \frac{1}{N} \sum_{n=1}^N \left(\frac{K_s^\dagger - K_s^{(n)}}{K_s^\dagger} \right)^2 \quad (9)$$

where $f_s^{(n)}$ is the average sampling frequency of the n -th sampling pattern. Since all the sampling patterns have the same time length τ , in practice it is usually more convenient to use the requested number of sampling points in a pattern K_s^\dagger and count the number of actual sampling points in a pattern $K_s^{(n)}$. This parameter is an average value of a relative frequency error of every sampling pattern. The lower the parameter e_f is, the better is the frequency stability of the generator. Additionally, let us introduce a γ_f parameter:

$$\gamma_f = \frac{1}{N} \sum_{n=1}^N \gamma_f^{(n)} \quad \gamma_f^{(n)} = \begin{cases} 0 & \text{for } K_s^\dagger = K_s^{(n)} \\ 1 & \text{for } K_s^\dagger \neq K_s^{(n)} \end{cases} \quad (10)$$

which is the ratio of patterns in a bag \mathbb{A} which violate the frequency stability requirement. The parameter $\gamma_f^{(n)} = 1$ denotes whether the average sampling frequency of the n -th pattern is incorrect.

3.2 Sampling point interval error parameters

Let us introduce statistical parameters which indicate how well the assessed generator meets the interval requirements discussed in Sec. 2.3.2. For a given n -th sampling pattern $\mathbb{T}^{(n)}$ let us create ordered subsets $\mathbb{D}_-^{(n)} \subset \mathbb{D}^{(n)}$ and $\mathbb{D}_+^{(n)} \subset \mathbb{D}^{(n)}$, where \mathbb{D} is a set with intervals between sampling points as in (6). These subsets contain intervals between samples which violate the minimum and the maximum requirements between sampling points t_{\min} and t_{\max} respectively:

$$\mathbb{D}_- = \{d_{-,k} \in \mathbb{D} : d_{-,k} < t_{\min}\} \quad (11)$$

$$\mathbb{D}_+ = \{d_{+,k} \in \mathbb{D} : d_{+,k} > t_{\max}\} \quad (12)$$

Now let us introduce statistical parameters e_{\min} and e_{\max} :

$$e_{\min} = \frac{1}{N} \sum_{n=1}^N (e_-^{(n)})^2 \quad e_-^{(n)} = \frac{|\mathbb{D}_-^{(n)}|}{|\mathbb{D}^{(n)}|} \quad (13)$$

$$e_{\max} = \frac{1}{N} \sum_{n=1}^N (e_+^{(n)})^2 \quad e_+^{(n)} = \frac{|\mathbb{D}_+^{(n)}|}{|\mathbb{D}^{(n)}|} \quad (14)$$

where $|\cdot|$ denotes the number of elements in a set (set's cardinality), and $|\mathbb{D}^{(n)}| = K_s - 1$ as in (6). These parameters contain the average squared ratio of the number of intervals in a pattern which violate minimum/maximum interval requirements to the number of all intervals between sampling points in a pattern. The lower the above parameters are, the better the evaluated generator meets interval requirements. Similarly to the frequency stability parameter, let us introduce γ_{\min} and γ_{\max} parameters:

$$\gamma_{\min} = \frac{1}{N} \sum_{n=1}^N \gamma_{\min}^{(n)} \quad \gamma_{\min}^{(n)} = \begin{cases} 0 & \text{for } |\mathbb{D}_-^{(n)}| = 0 \\ 1 & \text{for } |\mathbb{D}_-^{(n)}| > 0 \end{cases} \quad (15)$$

$$\gamma_{\max} = \frac{1}{N} \sum_{n=1}^N \gamma_{\max}^{(n)} \quad \gamma_{\max}^{(n)} = \begin{cases} 0 & \text{for } |\mathbb{D}_+^{(n)}| = 0 \\ 1 & \text{for } |\mathbb{D}_+^{(n)}| > 0 \end{cases} \quad (16)$$

which are additional parameters which are equal to ratios of patterns which violate minimum or maximum intervals between sampling patterns. Parameters $\gamma_{\min}^{(n)} = 1$ and $\gamma_{\max}^{(n)} = 1$ denote if the n -th pattern meets the requirement of minimum and maximum intervals respectively.

3.3 Ratio of incorrect patterns

It is possible to assign to every n -th pattern a parameter $\gamma^{(n)}$ which denotes if a pattern violates the frequency stability (2.3.1) or the interval requirements (2.3.2). The ratio of incorrect patterns γ of a bag \mathbb{A} is:

$$\gamma = \frac{1}{N} \sum_{n=1}^N \gamma^{(n)} \quad \gamma^{(n)} = \gamma_f^{(n)} \vee \gamma_{\min}^{(n)} \vee \gamma_{\max}^{(n)} \quad (17)$$

where \vee is a logical disjunction. Using parameter $\gamma^{(n)}$ it is possible to generate a sub-bag $\mathbb{A}^* \subseteq \mathbb{A}$ which contains only correct patterns from the bag \mathbb{A} :

$$\mathbb{A}^* = \{\mathbb{T} \text{ in } \mathbb{A} : \gamma^{(n)} = 0\} \quad (18)$$

where $\mathbb{T} \text{ in } \mathbb{A}$ signifies that a pattern \mathbb{T} is an element of a multiset \mathbb{A} . Please note that \mathbb{A} is a multiset, so patterns which are the elements of \mathbb{A} may be repeated, and patterns which are the elements of the multiset \mathbb{A}^* may also be repeated. Ideally, a sub-bag with correct patterns \mathbb{A}^* is identical to the original bag \mathbb{A} .

3.4 Quality parameter: Probability density function

Let us introduce a statistical parameter e_p which indicates whether the probability density of occurrence for grid points in patterns from bag \mathbb{A} is uniformly distributed:

$$e_p = \frac{1}{K_g} \sum_{m=1}^{K_g} (p_g(m) - 1)^2 \quad (19)$$

The probability of occurrence of the m -th grid point $p_g(m)$ is:

$$p_g(m) = \frac{K_g}{K_t} \sum_{n=1}^N g_m(n) \quad K_t = \sum_n K_s^{(n)} \quad (20)$$

where K_g is the number of sampling grid points in a sampling pattern, K_t is the total number of sampling points in all the patterns in a bag \mathbb{A} , and the parameter $g_m(n)$ indicates whether the m -th grid point is used in the n -th sampling pattern $\mathbb{T}^{(n)}$:

$$g_m(n) = \begin{cases} 0 & \text{if } mT_g \notin \mathbb{T}^{(n)} \\ 1 & \text{if } mT_g \in \mathbb{T}^{(n)} \end{cases} \quad (21)$$

Additionally, let us introduce a statistical parameter e_p^* which is calculated identically to e_p , but based on sampling patterns from subbag \mathbb{A}^* (18).

3.5 Quality parameter: Uniqueness of patterns

Let us create a set $\mathbb{A}_\#$ for a bag \mathbb{A} of N sampling patterns generated by the evaluated pattern generator which contains only unique patterns from \mathbb{A} . Similarly, let us create a set $\mathbb{A}_\#^*$ which contains only unique patterns from the subbag with correct patterns \mathbb{A}^* (18). Now let us introduce parameters η_N and η_N^* :

$$\eta_N = |\mathbb{A}_\#| \quad \eta_N^* = |\mathbb{A}_\#^*| \quad (22)$$

These parameters count the number of unique patterns and unique correct patterns in the bag \mathbb{A} with N generated patterns.

4 Pattern generators

Algorithms of sampling pattern generators are presented in this Section. Subsection 4.1 presents existed, widely known Jittered Sampling (JS) and Additive Random Sampling (ARS) algorithms. Subsection 4.2 presents the proposed sampling pattern generator algorithm, which is tailored to fulfill the requirements presented in 2.3 and 2.4. Please note that all the algorithms presented in this paper generate sampling patterns represented as indices of sampling grid points as in (5).

4.1 Jittered Sampling and Additive Random Sampling

Jittered Sampling and Additive Random Sampling algorithms are widely used to generate random sequences. There are 4 input variables to the JS and ARS algorithms: requested time of a sampling pattern τ , grid period T_g , requested average sampling frequency f_s^\dagger and the variance parameter σ^2 . The realizable time of a sampling pattern $\hat{\tau}$ may differ from the given requested time of a pattern τ if the given time is not a multiple of the given grid period T_g . Before

either of the algorithms is started, the number of grid points K_g in a sampling pattern, the realizable time of a sampling pattern $\hat{\tau}$ and the realizable requested number of sampling points \hat{K}_s^\dagger must be computed:

$$K_g = \left\lfloor \frac{\tau}{T_g} \right\rfloor \quad \hat{\tau} = K_g T_g \quad \hat{K}_s^\dagger = \lceil \hat{\tau} f_s^\dagger \rceil \quad (23)$$

where $\lfloor \cdot \rfloor$ signifies the rounding function, which returns an integer which is closest to the function's argument. Because the algorithms operate on a discrete set of grid points, the realizable requested average sampling frequency \hat{f}_s^\dagger may differ from the requested sampling frequency f_s^\dagger . The realizable requested average sampling frequency \hat{f}_s^\dagger and realizable requested average sampling period \hat{T}_s^\dagger is computed:

$$\hat{f}_s^\dagger = \frac{\hat{K}_s^\dagger}{\hat{\tau}} \quad \hat{T}_s^\dagger = \frac{1}{\hat{f}_s^\dagger} \quad \hat{N}_s^\dagger = \left\lceil \frac{\hat{T}_s^\dagger}{T_g} \right\rceil \quad (24)$$

where \hat{N}_s^\dagger is the requested average sampling period recalculated to the number of grid periods. If the computed realizable requested sampling frequency \hat{f}_s^\dagger is different from the requested sampling frequency f_s^\dagger , the problem of generation of sampling patterns is not well stated. Before the algorithms start, the index of a correct sampling point \hat{k} and the starting position of the sampling point n_0 must be reset:

$$\hat{k} = 0 \quad n_0 = 0 \quad (25)$$

In the JS algorithm, every sampling point is a uniform sampling point which is randomly "jittered":

$$n_{JS,k}^* = \lceil k \hat{N}_s^\dagger + \sqrt{\sigma^2} x_k \hat{N}_s^\dagger \rceil \quad x_k \sim \mathcal{N}(0, 1) \quad (26)$$

where $\mathcal{N}(0, 1)$ denotes a standard normal distribution. In the ARS algorithm every sampling point is computed using the previous sampling point to which an average sampling period and a random value are added:

$$n_{ARS,k}^* = \lceil n_{k-1} + \hat{N}_s^\dagger + \sqrt{\sigma^2} x_k \hat{N}_s^\dagger \rceil \quad x_k \sim \mathcal{N}(0, 1) \quad (27)$$

Fig. 4 illustrates generation of sampling patterns in the JS and ARS algorithms.

The practical versions of both JS and ARS algorithms are presented in Alg. 1. After generation of a pattern, any repeated sampling point must be removed (line 12 of Alg. 1). It is because in these algorithms there is no guarantee that sampling points are not repeated.

4.2 'ANGIE' algorithm

We propose an algorithm which would perfectly cope with the requirements described in 2.3 and as much as possible with the requirements in 2.4. The ratio of incorrect patterns γ (17) generated by the algorithm should always equal 0, while keeping the probability density parameter e_p (Sec. 3.4) as low as possible

Algorithm 1 JS and ARS algorithms - pseudo code

```

1: function  $[\mathbb{T}] = \text{JS/ARS}(\tau, T_g, f_s^\dagger, \sigma^2)$ 
2: Compute  $K_g, \hat{\tau}$  and  $\hat{K}_s^\dagger$  as in (23)
3: Compute  $\hat{f}_s^\dagger, \hat{T}_s^\dagger$  and  $\hat{N}_s^\dagger$  as in (24)
4: Reset  $\hat{k}$  and  $n_0$  as in (25)
5: FOR  $k = 1$  TO  $\hat{K}_s^\dagger$ 
6:   Draw sampling moment  $n_{\text{JS},k}^*$  (26) or  $n_{\text{ARS},k}^*$  (27)
7:   IF  $n_k^* > 0$  AND  $n_k^* < \hat{\tau}$ 
8:      $n_{\hat{k}} \leftarrow n_k^*$ 
9:     Assign  $\mathbb{T}'(\hat{k}) \leftarrow n_{\hat{k}}$ 
10:     $\hat{k} \leftarrow \hat{k} + 1$ 
11: END
12: Remove repeated sampling points in  $\mathbb{T}$ 

```

and the uniqueness parameter $\eta_N = \eta_N^*$ (Sec. 3.5) as high as possible. The parameters e_p^* and η_N^* must equal e_p and η_N respectively, as the subbag with correct patterns \mathbb{A}^* must be identical to the subbag with all the patterns \mathbb{A} (all the generated patterns must be correct). Therefore we propose the rANdOm sampling Generator with Intervals Enabled (ANGIE) algorithm. The input variables to the algorithm are identical to the JS and ARS algorithms (4.1), with additional variables for the allowed time between samples (t_{\min}, t_{\max}).

Before the ANGIE algorithm starts, the following precomputations must be done. Similarly to the JS and ARS algorithms, the number of grid points in a sampling pattern (K_g), the realizable time of a sampling pattern ($\hat{\tau}$) and the realizable number of sampling points in a sampling pattern \hat{K}_s^\dagger must be computed as in (23). Then the minimum and the maximum time between sampling points must be recalculated to the number of grid points:

$$K_{\min} = \left\lceil \frac{t_{\min}}{T_g} \right\rceil \quad K_{\max} = \left\lfloor \frac{t_{\max}}{T_g} \right\rfloor \quad (28)$$

where $\lceil \cdot \rceil$ signifies the ceiling function which returns the lowest integer which is higher or equal to the function's argument. In the proposed algorithm there are 2 limit variables, n_k^- and n_k^+ , which are the first and the last possible position of a k -th sampling point. These variables are updated after generation of every sampling point. Before the algorithm starts these variables must be initialized:

$$n_1^- = 1 \quad n_1^+ = K_g - K_{\min}(\hat{K}_s^\dagger - 1) \quad (29)$$

The number of sampling points left to be generated is updated before generation of every sampling point:

$$n_k^{\text{left}} = \hat{K}_s^\dagger - k + 1 \quad (30)$$

where k is the index of the current sampling point. The average sampling period for the remaining n_k^{left} sampling points and the expected position e_k of the k -th

sampling point is:

$$e_k = n_{k-1} + n_k^\dagger \quad n_k^\dagger = \left\lceil \frac{K_g - n_{k-1}}{n_k^{\text{left}} + 1} \right\rceil \quad (31)$$

In the proposed algorithm, a k -th sampling point n_k may differ from its expected position e_k by the interval n_k^d . Before computing this interval the algorithm must compute intervals to the limits:

$$n_k^{d-} = |e_k - n_k^-| \quad n_k^{d+} = |n_k^+ - e_k| \quad (32)$$

and then the lower from the above intervals is the correct interval n_k^d :

$$n_k^d = \min(n_k^{d-}, n_k^{d+}) \quad (33)$$

The first sampling point is drawn using a uniformly distributed variable x^u :

$$n_1 = \lceil x_1^u n_1^\dagger \rceil \quad x_1^u \sim \mathcal{U}(0, 1) \quad (34)$$

while the rest of the sampling points are drawn using the normal distribution:

$$n_k = e_k + [x_k n_k^d] \quad x_k \sim \mathcal{N}(0, \sigma^2) \quad (35)$$

Finally, the algorithm checks whether the drawn sampling moment n_k violates the limits n_k^- and n_k^+ :

$$n_k = \begin{cases} n_k^+ & \text{for } n_k > n_k^+ \\ n_k^- & \text{for } n_k < n_k^- \end{cases} \quad (36)$$

In the last step the limits for the next sampling point are computed. The lower and the higher limits are computed as:

$$n_{k+1}^- = n_k + K_{\min} \quad n_{k+1}^+ = K_g - K_{\min}(n_k^{\text{left}} - 2) \quad (37)$$

If the maximum time between samples is valid ($t_{\max} < \text{inf}$), then the higher limit should be additionally checked for t_{\max} :

$$n_{k+1}^+ = \min(n_{k+1}^+, n_k + K_{\max}) \quad (38)$$

The proposed algorithm is presented in Alg. 2.

5 Numerical experiment

In this section, the performance of the proposed ANGIE algorithm is experimentally compared with the JS and ARS algorithms. A toolbox with pattern generators and evaluation functions was created to facilitate the experiment. Emphasis was set on validation of parts of the software. The toolbox, together with its documentation, is available online at [21]. Using the content available at [21] it is possible to reproduce the presented numerical simulations.

Algorithm 2 'ANGIE' algorithm - pseudo code

```
1: function  $[\mathbb{T}] = \text{ANGIE}(\tau, T_g, f_s^\dagger, t_{\min}, t_{\max}, \sigma^2)$ 
2: Compute  $K_g, \hat{\tau}$  and  $\hat{K}_s^\dagger$  as in (23)
3: Compute  $K_{\min}$  and  $K_{\max}$  as in (28)
4: Initialize the limits  $n_1^-$  and  $n_1^+$  as in (29)
5: FOR  $k = 1$  TO  $\hat{K}_s^\dagger$ 
6:   Update the number of sampling points left  $n_k^{\text{left}}$  as in (30)
7:   Compute the expected position  $e_k$  as in (31)
8:   Compute the interval  $n_k^d$  as in (33)
9:   Draw sampling moment  $n_k$  as in (34) or (35)
10:  Check and correct  $n_k$  as in (36)
11:  Assign  $\mathbb{T}'(k) \leftarrow n_k$ 
12:  Update the limits  $n_{k+1}^-$  and  $n_{k+1}^+$  as in (37) and (38)
13: END
```

5.1 Experiment #1 - setup

The duration τ of sampling patterns is set to 1 ms, sampling grid period T_g is equal to 1 μs . The requested average sampling frequency of patterns is set to 100 kHz, which corresponds to an average sampling period equal to 10 μs . The minimum time between sampling points is $t_{\min} = 5\mu\text{s}$, and there is no requirement for maximum time between sampling points ($t_{\max} = \text{inf}$). The variance σ^2 is logarithmically swept in the range $[10^{-4}, 10^2]$.

The computed statistical parameters of sampling patterns are automatically tested for convergence. A mean value is accounted as converged, if for the last $2 \cdot 10^4$ patterns it did not change more than 1% of the mean value computed for all the patterns currently tested. The minimum number of sampling patterns tested is 10^5 . The uniqueness parameters η_N and η_N^* (22) are computed after $N = 10^5$ patterns.

5.2 Experiment #1 - results

Error parameters computed for the tested sampling pattern generators are plotted in Fig. 7. The ratio of incorrect patterns are plotted in Fig. 6. This ratio for the ANGIE algorithm (blue \diamond) is equal to 0 for all the values of variance σ^2 . Thus, all the patterns have correct average sampling frequency and intervals between sampling points. Patterns generated by the JS (green \blacktriangledown) and the ARS algorithms (black \blacktriangle) are all correct for very low values of the variance σ^2 , but the quality parameters e_p and η_{10^5} for these σ^2 values are poor (Fig. 8 and Fig. 9). In Fig. 7 it can be seen that for nearly all the values of variance σ , the frequency stability of the patterns generated by the JS and the ARS algorithms is compromised, and for most of the values of σ^2 , the requirement of minimum intervals between sampling points is not met by these algorithms.

The best values of the parameter e_p are achieved for the JS (green \blacksquare) and the ARS (black \blacksquare) algorithms (Fig. 8), but only if all the patterns (also incorrect)

are taken into account (parameter e_p). If the quality parameter was computed only for the correct patterns (parameter e_p^*), it can be clearly seen that the proposed algorithm (blue ■) performs significantly better than the JS (yellow ○) and the ARS algorithms (yellow ★). Furthermore, the best values of e_p^* are found for the values of variance σ for which most of the patterns produced by the JS and the ARS algorithms are incorrect. Plots of the best probability density functions found for the tested algorithms are in Fig. 10.

Fig. 9 shows the number of unique patterns produced by the tested algorithms. The number of unique correct patterns produced by the proposed algorithm is higher than the number produced by the JS and the ARS algorithms for any variance value $\sigma^2 \geq 10^{-2}$.

The above results show that the proposed algorithm ANGIE performs better than the JS and the ARS algorithms. All the patterns generated by the ANGIE algorithm are correct, have a parameter $\gamma^{(n)}$ defined as in (17) equal to 0. The quality parameters described in Sec. 3.4 and Sec. 3.5 are better for the proposed algorithm. It can be seen that the variance value σ^2 , which is an internal algorithm parameter, should be adjusted to a given problem. For the given problem, the proposed algorithm performs best for $\sigma^2 = 10^{-2}$.

5.3 Experiment #2 - setup

In the second experiment four different cases (A-D) of sampling patterns are studied. Parameters of these cases are collected in Table 1. In the first two cases there are requirements of both the minimum and the maximum distance between sampling points. In the second case there are only 5 sampling points requested p. sampling pattern, and the number of sampling grid points is limited to 100. In the third case there are no requirements imposed on distances between sampling points, so there is only the requirement of stable average sampling frequency. This case is distinctive from others, because the number of sampling points p. sampling pattern is high (10^4), and the grid period is very low. In the last case there is a requirement of the maximum distance between sampling points. In all the four cases the variance σ^2 is logarithmically swept in the range $[10^{-4}, 10^2]$.

In this experiment there are three quality parameters measured for all the three generators (JS, ARS and ANGIE). The first parameter is the ratio of incorrect patterns γ (17). The second is the probability density parameter e_p^* as in (19), but computed only for the correct patterns. The third quality parameter is the number of unique correct patterns in the first 10^4 generated patterns $\eta_{10^4}^*$ (22).

5.4 Experiment #2 - results

Results of this experiment are shown on Figures 11–13. Each Figure presents a measured quality parameter for all the four cases. The ratio of incorrect patterns γ is on Fig. 11, the probability density parameter e_p^* is on Fig. 12, and the number of unique correct patterns $\eta_{10^4}^*$ is on Fig. 13.

case	Independent parameters					Dependent parameters			
	τ [ms]	T_g [μs]	f_s [kHz]	t_{\min} [ms]	t_{\max} [ms]	K_g	\hat{K}_s^\dagger	K_{\min}	K_{\max}
A	10^3	10^3	0.05	10	30	10^3	50	10	30
B	0.1	1	50	0.015	0.028	100	5	15	28
C	10^3	1	10	—	—	10^6	10^4	—	—
D	0.005	$25 \cdot 10^{-5}$	10^5	—	$14 \cdot 10^{-6}$	$2 \cdot 10^4$	500	—	56

Table 1: Parameters of sampling patterns used in all the four cases of experiment #2. Independent parameters are: time length of sampling patterns (τ), grid period (T_g), requested average sampling frequency (f_s), minimum allowed time between sampling points (t_{\min}), maximum allowed time between sampling points (t_{\max}). Shown dependent parameters are: the number of grid points (K_g), the requested realizable number of sampling points (\hat{K}_s^\dagger), the minimum and maximum time between the sampling points recalculated to the number of grid points (K_{\min} and K_{\max}).

Let us take a look at the ratio of incorrect patterns (Fig. 11). The ANGIE algorithm generates only correct sampling patterns. Hence to line 10 in the algorithm (see Algorithm 2), the minimum and the maximum distances between sampling points are kept. Lines 6–8 in the ANGIE algorithm ensure that there will be place for the correct number of sampling points in all the generated sampling patterns. To the contrary, both ARS and JS algorithms generate a lot of incorrect patterns. For the high values of variance σ^2 there are only incorrect patterns generated by these two algorithms.

In the three cases (A, C, D) the best probability density parameter e_p^* (Fig. 12) measured for patterns generated by the ANGIE algorithm is better than for the other two algorithms. Additionally, it can be seen in Fig. 13 that the generated number of unique correct sampling patterns is in all the four cases significantly higher for the proposed ANGIE algorithm. Let us take a closer look on the case B. In this case, the best probability density parameter e_p^* found for the algorithm ARS ($\sigma^2 = 10^{-0.5}$) is slightly better than the best e_p^* found for the ANGIE ($\sigma^2 = 10^{1.5}$). Still, the number of unique patterns is significantly better for the above values of σ^2 for ANGIE algorithm, and very most of the patterns generated by the ARS are incorrect for $\sigma^2 = 10^{-0.5}$.

We tried to find a case for which ARS and JS algorithms would clearly and distinctly outperform the ANGIE, but it turned out to be an impossible task. Still though, it is difficult to provide the reader with one gold rule which algorithm should be used. In practical applications there may be a huge number of different sampling scenarios, in this paper we covered only a tiny fraction of examples, and therefore every case should be considered separately. In general, ANGIE algorithm will always generate correct sampling patterns. But if these sampling patterns will have all quality parameters (especially e_p^*) better than sampling patterns generated by the other algorithms, that is an another issue. From our experience we claim that indeed, in most of the cases ANGIE is

the right choice. However, there might be applications in which, for example, equi-probability of occurrence of every sampling point is a critical matter and other algorithms might perform better. In practical applications, a numerical experiment should be always conducted to choose a correct pattern generator and to adjust variance value σ^2 .

We prepared a software PATterns TESting System (PATES), which is open-source and available online [21]. This software contains all the three generators considered in this paper plus routines which compute the proposed quality parameters. With this software a user is able to test the generators for his own sampling scenario. We have created a graphical user interface to the software (Fig. 14), which makes using the system more intuitive. Reproducible research scripts which can be used to produce results from the presented experiments are also available in [21].

6 Implementation issues

In this Section we discuss some of the implementation issues of random sampling patterns. In this paper, we focus on offline sampling pattern generation (Fig. 5), where patterns are prepared offline by a computational server and then stored in a memory which is a part of a signal processing system. Immediate generation of sampling patterns would require very fast pattern generators which are able to generate every sampling point in a time much shorter than minimum time between sampling points t_{\min} . The ANGIE algorithm (Alg. 2) requires a number of floating point computations before every sampling point is computed, therefore very powerful computational circuit would be necessary in real time applications where $t_{\min} < 1\mu s$.

6.1 Software patterns generator

In practical applications there is a need to generate $N \gg 1$ sampling patterns. Sampling patterns are generated offline (Fig. 5) on a computational server. In naive implementation, Alg. 2 is repeated N times to generate N random sampling patterns. This approach is suboptimal, because computation of initial parameters from equations (23) and (28) (lines 2-3) is unnecessarily repeated N times. In the optimal implementation lines 2-3 are performed only once before a bag of patterns is generated.

We implemented the ANGIE algorithm (naive implementation) in Python. Furthermore, we prepared an implementation in C and an optimized implementation in Python (vectorized code). All the implementations are available for download at [21]. Fig. 15 shows time needed to generate $N = 10^5$ sampling patterns. Parameters of sampling patterns are identical to the parameters used in the experiment described in Section 5.1. The average sampling frequency is swept from 10 kHz to 100 kHz, and the duration of the patterns is kept fixed. Measurements were made on an Intel Core i5-3570K CPU, and a single core of the CPU was used.

The ANGIE algorithm operates mostly on integer numbers, and therefore it requires maximally only three floating point operations p. sampling point. The algorithm time complexity vs. the average sampling frequency of a pattern is $O(n)$ (consider the logarithmic vertical scale), because lines 5-13 in Alg. 2 are repeated for every sampling point which must be generated. As expected, the optimized vectorized Python / optimized C implementation is much faster than the naive Python implementation.

6.2 Driver of an analog-to-digital converter

The analog-to-digital converter (ADC) driver is a digital circuit which triggers the converter according to a given sampling pattern. The maximum clock frequency of the driver determines the minimum grid period. Detailed construction of the driver depends on the used ADC because the driver must generate specific signals which drive the ADC.

A simple driver marks the 'sample now' signal every time the grid counter reaches a value equal to the current sampling time point. Such a driver was implemented in VHDL language. The structure of the driver is shown in Fig. 16. Due to the internal structure of the control circuit, the grid period is eight times longer than the input clock period. Table 2 contains results of synthesis of the driver in four different Xilinx FPGAs.

Xilinx FPGA	Max clock frequency [MHz]	Min grid period T_g [ns]
Spartan 3	439.97	18.2
Virtex 6	1078.98	7.4
Artix 7	944.47	8.5
Zynq 7020	1160.36	6.9

Table 2: Maximum clock values and minimum grid periods of an implemented driver in different Xilinx FPGAs

Sampling patterns are read from a ROM. The amount of memory n_m used to store a sampling pattern [in bytes] is:

$$n_m = K_s \cdot \left\lceil \frac{\log_2 K_g}{8} \right\rceil \quad (39)$$

where K_g is the number of grid points in a pattern and K_s is the number of sampling points in a pattern. Depending on the available size of memory, different numbers of sampling patterns can be stored. Fig. 17 shows the relation between the memory size and the probability density parameter e_p (19) computed for the proposed ANGIE algorithm. The parameters of the sampling patterns are identical to the parameters used in the experiment described in Section 5.1, although four different average sampling frequencies are used.

As expected, the higher the average sampling frequency of patterns, the better the distribution of probability density function (parameter e_p is lower).

The higher the average sampling frequency of patterns, the more the memory needed to achieve the best possible probability density parameter e_p . If the available memory is low, the probability density function becomes less equiprobable.

7 Conclusions

This paper discussed generation of random sampling patterns dedicated to event-driven ADCs. Constraints and requirements for random sampling patterns and pattern generators were discussed. Statistical parameters which evaluate sampling pattern generators were introduced. We proposed a new algorithm which generates constrained random sampling patterns. The patterns generated by the proposed algorithm were compared with patterns generated by the state-of-the-art algorithms (Jittered Sampling and Additive Random Sampling). It was shown, that the proposed algorithm performs better in generation of random sampling patterns dedicated to event-driven ADCs. Implementation issues of the proposed method were discussed.

References

- [1] H. S. Shapiro, R. A. Silverman, "Alias free sampling of Random Noise", *IEEE Trans. Info. Theory*, vol. 16, pp. 147–152, (1960)
- [2] H. G. Feichtinger and K. Grchenig and T. Strohmer, "Efficient Numerical Methods In Non-Uniform Sampling Theory", *Numerische Matematik*, vol. 69, pp. 423–440, (1995)
- [3] I. Homjakovs, M. Greitans, R. Shavelis, "Real-Time Acquisitions of Wide-band Signals Data Using Non-Uniform Sampling", *Proc. IEEE Eurocon 2009*, pp. 1158–1163, Saint-Petersburg, May. 2009.
- [4] Hui-Qing Liu, "ADS82x ADC with non-uniform sampling clock.", *Analog Applications Journal*, (2005)
- [5] M. Wakin, S. Becker, E. Nakamura, M. Grant, E. Sovero, D.Ching, J. Yoo, J. Romberg, A. Emami-Neyestanak, E. Candes, "A Nonuniform Sampler for Wideband Spectrally-Sparse Environments", *IEEE Trans. Emerg. Sel. Topics Circuits Syst.* vol. 2(3), (2012)
- [6] E.J. Candès and M. B. Wakin, "An Introduction To Compressive Sampling", *IEEE Signal Process. Mag.*, vol. 25(2), pp. 21–30, (2008)
- [7] J. Laska, S. Kirolos, Y. Massoud, R. Baraniuk, A. Gilbert, M. Iwen and M. Strauss, "Random Sampling for Analog-to-Information Conversion of Wide-band Signals", *Proc. IEEE Dallas Circuits and Systems Workshop (DCAS)*, pp. 119–122, Dallas, USA, (2006)
- [8] R.G. Baraniuk, M. Davenport, R. Devore, M. Wakin, "A Simple Proof of the Restricted Isometry Property for Random Matrices", *Constructive Approximation*, vol. 28(3), pp. 253–263, (2008)
- [9] Analog Devices (2013). "A/D Converters. Analog Devices.",[Online] Available: <http://www.analog.com/en/analog-to-digital-converters/ad-converters/products/index.html>
- [10] B. Le, T. W. Rondeau, J. H. Reed, W. Bostian, "Analog-to-Digital Converters. A review of the past, present, and future.", *IEEE Sig. Proc. Mag.*, vol. 22(6), (2005)
- [11] Xilinx (2012). "7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter",[Online] Available: http://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf
- [12] F. Marvasti, "Spectral analysis of irregular samples of multidimensional signals," , *Presented the 6th Workshop on Multidim. Signal Processing*, Pacific Grove, California, (sep. 1989)

- [13] A. C. Gilbert, M. J. Strauss, and J. A. Tropp, “A Tutorial on Fast Fourier Sampling”, *IEEE Signal Process. Mag.*, vol. 25, pp. 57-66, (2008)
- [14] F. Marvasti, “Nonuniform Sampling, Theory and Practice”, *Springer Science + Business Media*, (2001), ISBN: 978-1-4613-5451-2, New York, USA
- [15] J.J. Wojtiuk, “Randomized Sampling for Radio Design”, *PhD Thesis*, University of Southern Australia, (2000)
- [16] Y. P. Lin, and P.P. Vaidyanathan, “Periodically Nonuniform Sampling of Bandpass Signals”, *IEEE Trans. Circuits Syst. II*, vol. 45(3), pp. 340–351, (1998)
- [17] I. Bilinskis, A. Mikelsons, “Randomized Signal Processing.”, *Prentice Hall*, (1992), ISBN: 978-0-137-51074-0, Cambridge, UK
- [18] F. Papenfuß, Y.Artyukh, E. Boole, D. Timmermann, “Nonuniform Sampling Driver Design For Optimal ADC Utilization”, *Proc. Internal Symposium on Circuits and Systems, 2003 (ISCAS'03)*, vol. 4, pp. 516–519, Bangkok, Thailand, (2003)
- [19] M. Ben-Romdhane, C. Rebai, P. Desgreys, A. Ghazeli, P. Loumeau, “Pseudorandom Clock Signal Generation for Data Conversion in a Multistandard Receiver”, *Proc. IEEE Int. Conference on Design and Technology of Integrated Systems in Nanoscale Era*, pp. 1-4, Tozeur, Tunisia, (2008)
- [20] P. Vandewalle, J. Kovacevic, and M. Vetterli, “Reproducible Research in Signal Processing [What, why and how]”, *IEEE Signal Process. Mag.*, vol. 26(3), pp. 37–47, (2009)
- [21] Aalborg University (2013). “IRfDUCS project”, [Online] Available: <http://www.irfducs.org/pates/>

Acknowledgment

The work is supported by The Danish Council for Independent Research under grant number 0602–02565B.

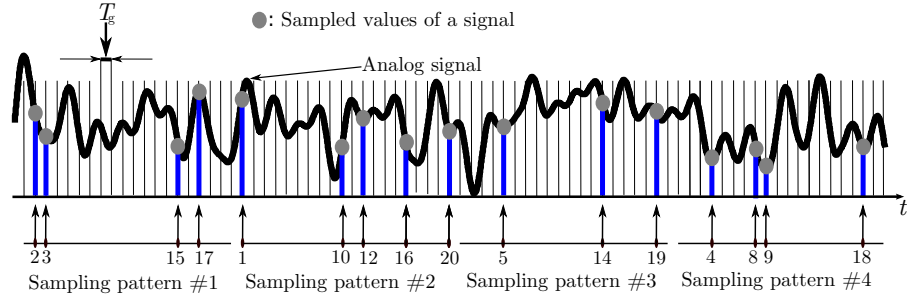


Figure 1: Example of unconstrained random sampling patterns applied to an analog signal. There is no minimum nor maximum allowed interval between sampling points. Furthermore, patterns contain different number of sampling points.

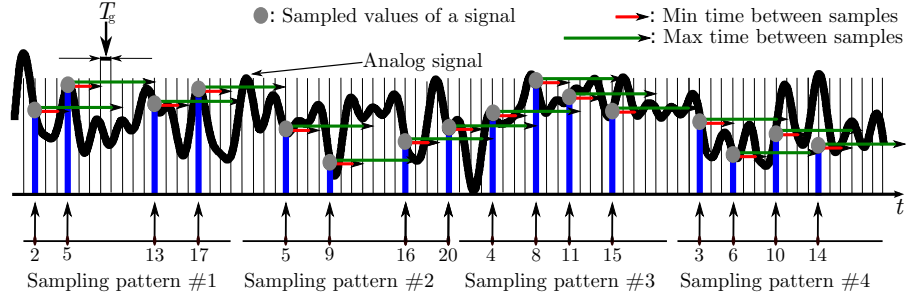


Figure 2: Example of constrained random sampling patterns applied to an analog signal. There is a minimum (red arrow) and maximum (green arrow) allowed interval between sampling points. Furthermore, every pattern has the equal number of sampling points.

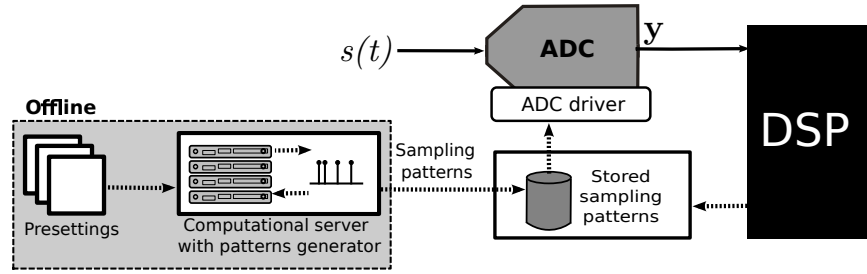


Figure 3: Offline generation of sampling patterns. Sampling patterns are prepared offline on a computational server, and then stored in a memory in the sampling system.

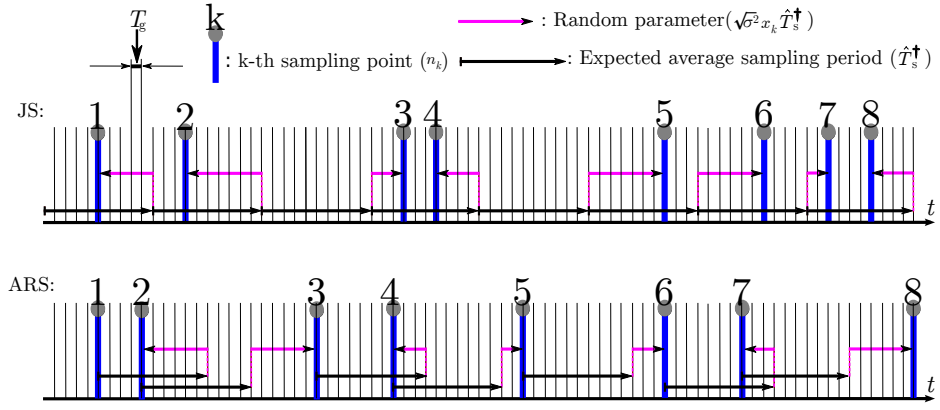


Figure 4: Illustration of generation of sampling patterns in Jittered Sampling (JS) and Additive Random Sampling (ARS) algorithms.

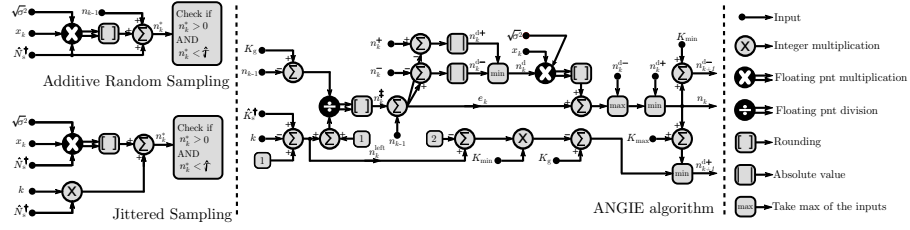


Figure 5: Block diagram showing the generation of one sampling point in the Additive Random Sampling, the Jittered Sampling and the ANGIE algorithm.

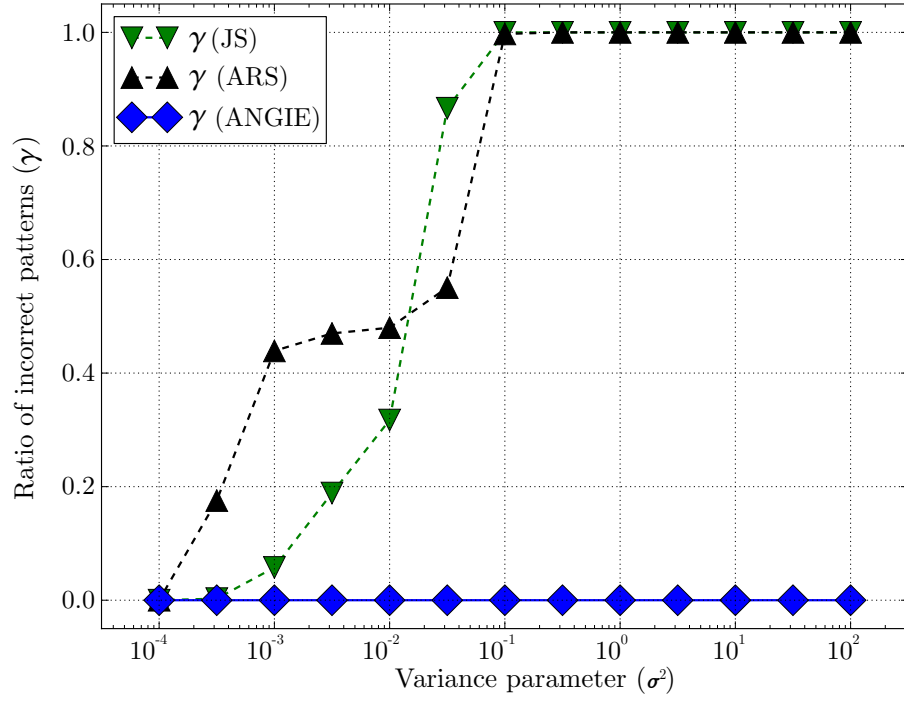


Figure 6: Ratio of incorrect patterns γ computed for patterns generated by the JS, ARS and ANGIE algorithms (experiment #1).

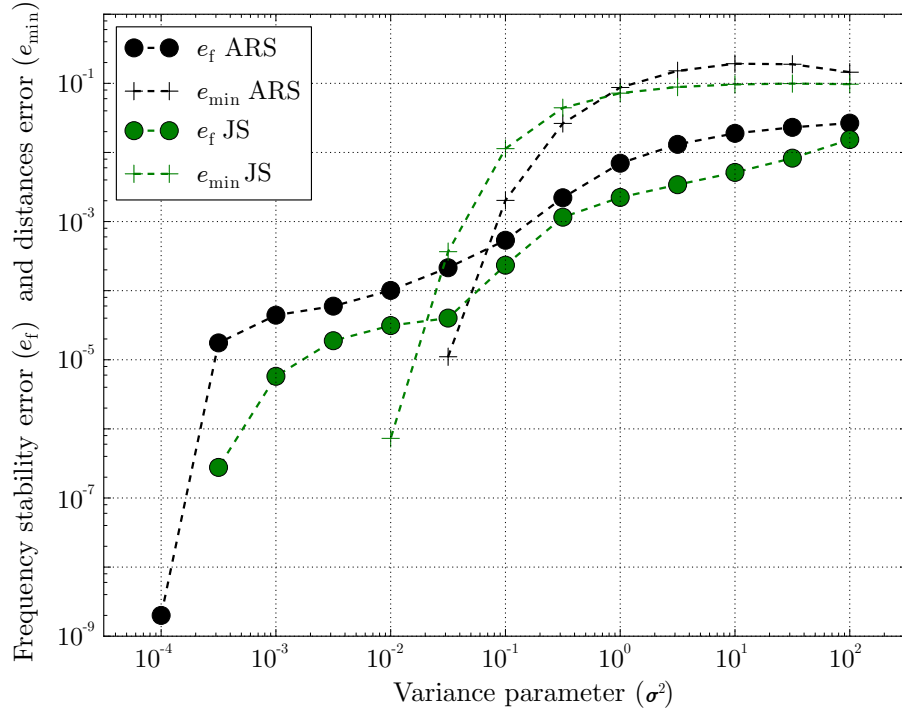


Figure 7: Frequency stability error e_f and intervals error e_{\min} computed for patterns generated by the JS and ARS algorithms (experiment #1). The error parameters are not plotted for the ANGIE algorithm because errors for this algorithm are equal 0 (all the patterns generated by the algorithm are correct - Fig. 6).

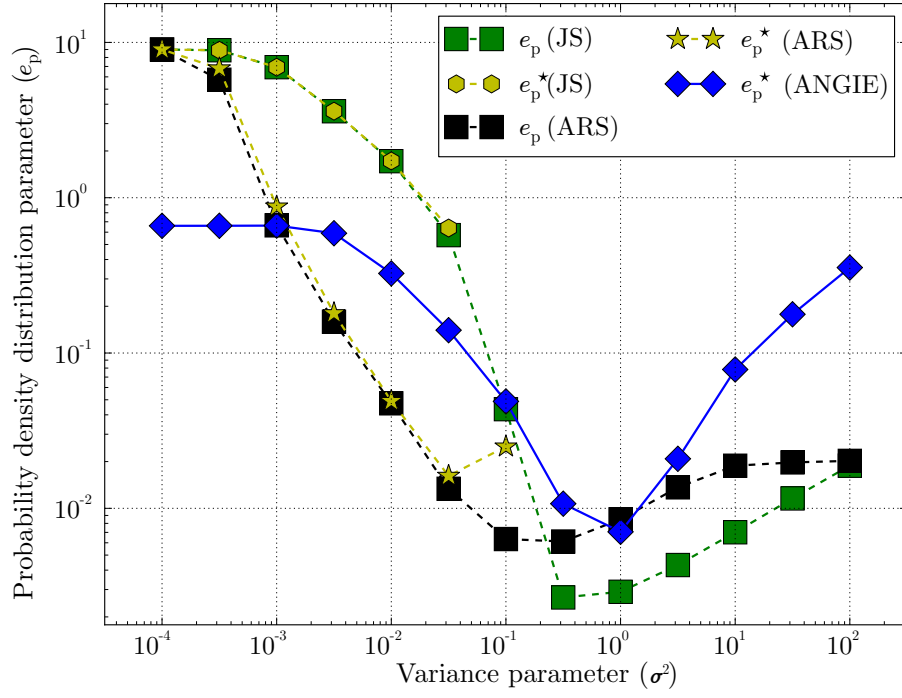


Figure 8: Probability density parameter e_p computed for patterns generated by the JS, ARS and ANGIE algorithms (experiment #1).

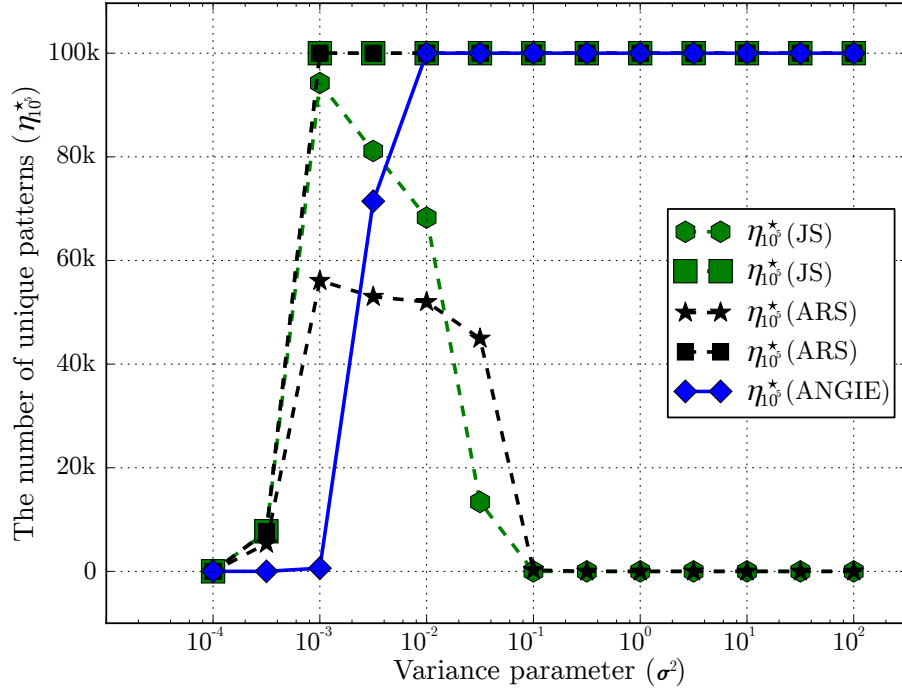


Figure 9: The number of unique patterns η_{10^5} computed for patterns generated by the JS, ARS and ANGIE algorithms (experiment #1). The parameter $\eta_{10^5}^*$ is not plotted for the ANGIE algorithm since it is equal to the parameter η_{10^5} for this algorithm. It is because the subbag $\mathbb{A}^* = \mathbb{A}$ for the ANGIE algorithm (all the patterns generated by the algorithm are correct - ref. to Fig. 6)

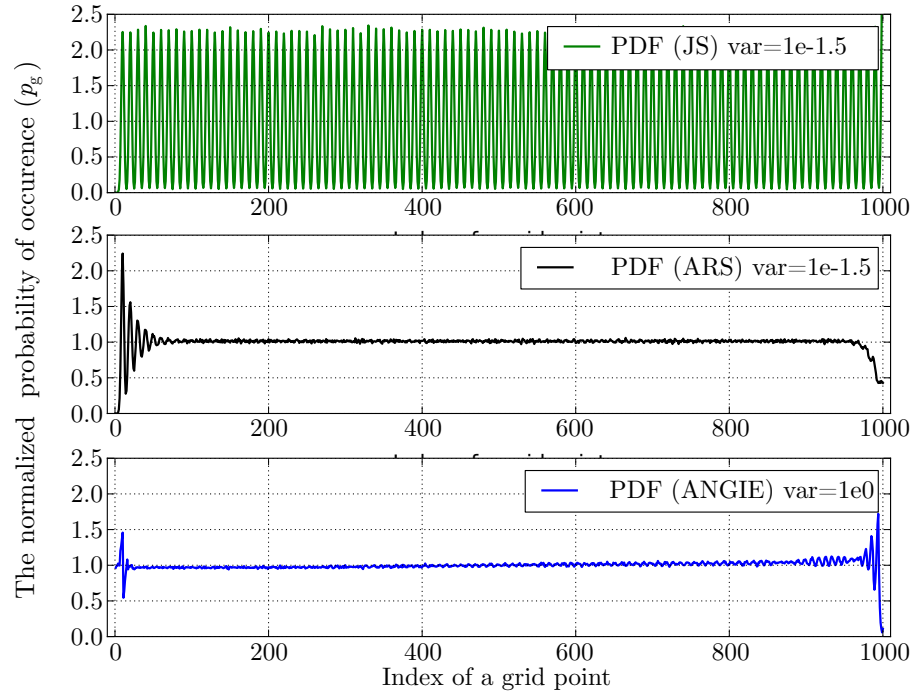


Figure 10: The best probability density functions of grid points found for the tested sampling pattern generators (experiment #1).

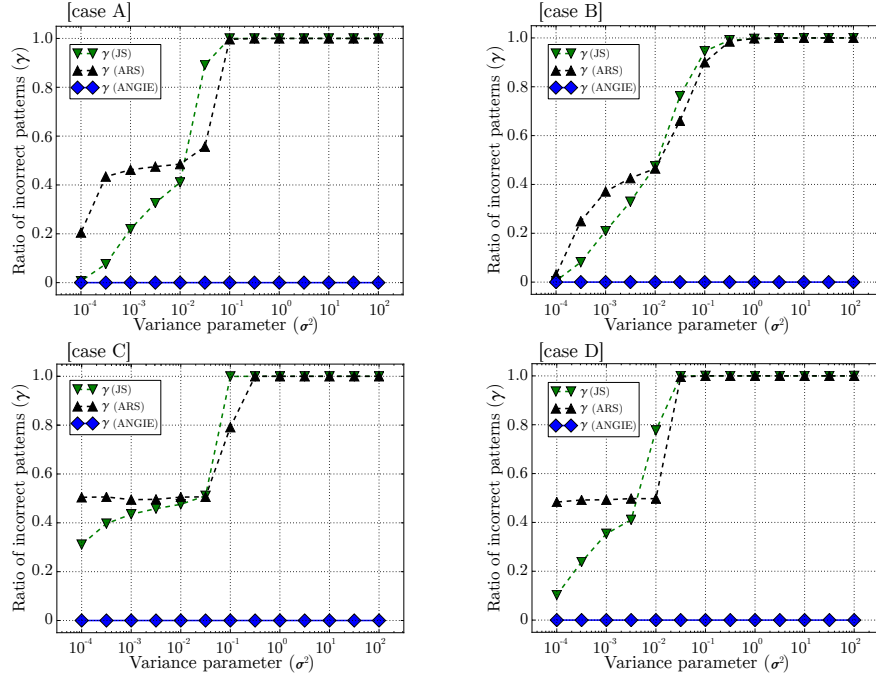


Figure 11: Ratio of incorrect patterns γ computed for patterns generated by the JS, ARS and ANGIE algorithms in all the four cases of the experiment #2.

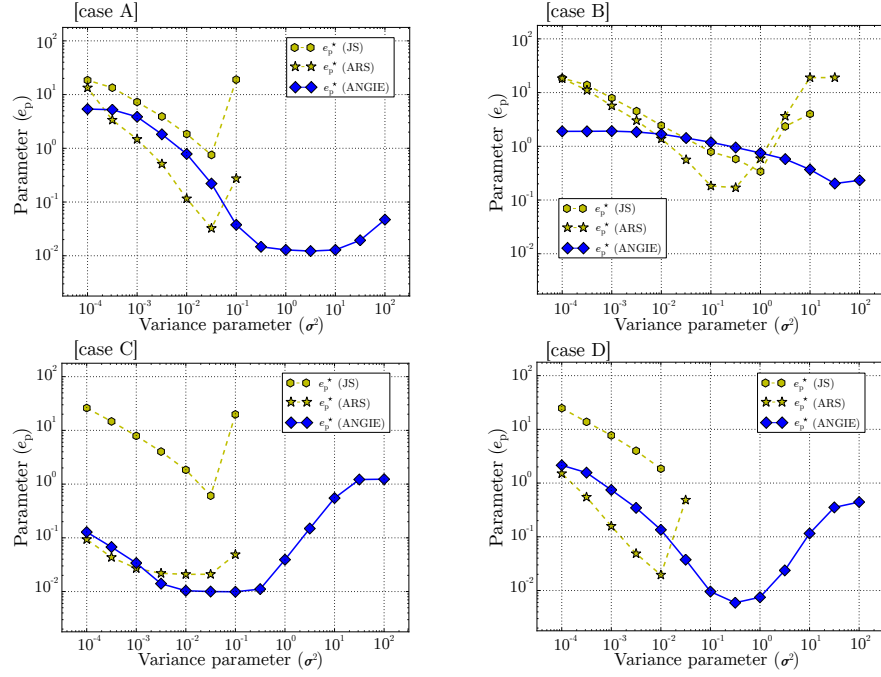


Figure 12: Probability density parameter e_p^* (parameter computed for correct patterns only) computed for patterns generated by the JS, ARS and ANGIE algorithms in all the four cases of the experiment #2.

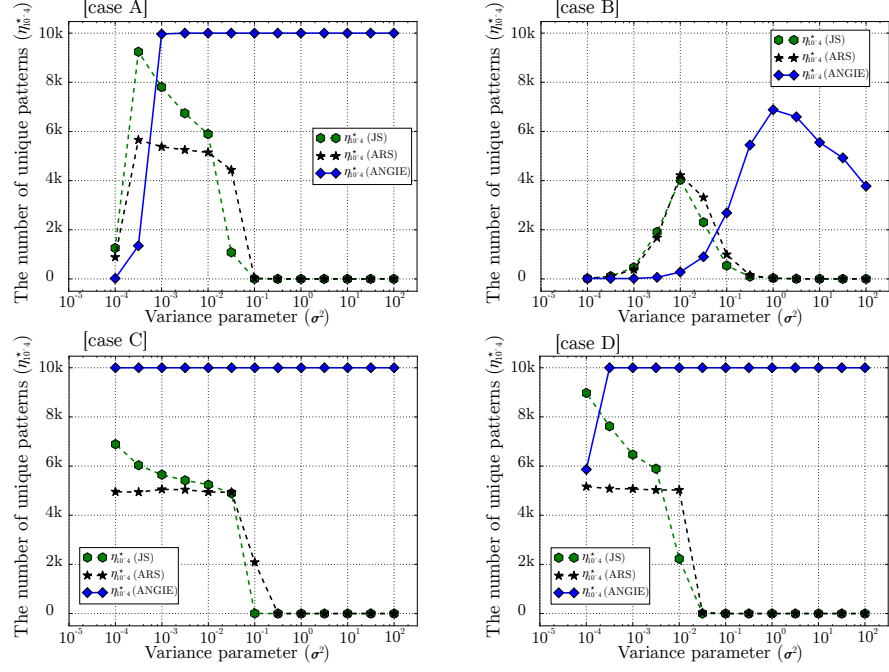


Figure 13: The number of unique patterns $\eta_{10^4}^*$ (parameter computed for correct patterns only) computed for patterns generated by the JS, ARS and ANGIE algorithms in all the four cases of the experiment #2.

Project names: <input type="text" value="smp_comparison"/>		Patterns storing Directory for patterns: <input type="text" value="patterns_storage"/> <input type="button" value="Browse"/>		Stop conditions: Min no. of patterns analyzed: <input type="text" value="10"/> K Max no. of patterns analyzed: <input type="text" value="100"/> K Max time: <input type="text" value="3600"/> s Consecutive convergences needed to stop: <input type="text" value="1"/>	
Available RAM: <input type="text" value="1000"/> MB Max size of a patterns pack: <input type="text" value="100"/> K					
Pattern generator #1: <input checked="" type="checkbox"/> Variance sweep Start: <input type="text" value="100"/> Product: <input type="text" value="0.3162"/> Points: <input type="text" value="13"/> Type: <input type="text" value="US"/> Time: <input type="text" value="100"/> us Grid: <input type="text" value="1"/> us Freq: <input type="text" value="50.0"/> kHz Var: <input type="text" value="1.0"/> <input type="checkbox"/> Min d: <input type="text" value="1.0"/> us <input type="checkbox"/> Max d: <input type="text" value="10.0"/> us		Pattern generator #2: <input checked="" type="checkbox"/> Generator on <input checked="" type="checkbox"/> Variance sweep Start: <input type="text" value="100"/> Product: <input type="text" value="0.3162"/> Points: <input type="text" value="13"/> Type: <input type="text" value="ARS"/> Time: <input type="text" value="100"/> us Grid: <input type="text" value="1"/> us Freq: <input type="text" value="50.0"/> kHz Var: <input type="text" value="1.0"/> <input type="checkbox"/> Min d: <input type="text" value="1.0"/> us <input type="checkbox"/> Max d: <input type="text" value="10.0"/> us		Pattern generator #3: <input checked="" type="checkbox"/> Generator on <input checked="" type="checkbox"/> Variance sweep Start: <input type="text" value="100"/> Product: <input type="text" value="0.3162"/> Points: <input type="text" value="13"/> Type: <input type="text" value="ANGIE"/> Time: <input type="text" value="100"/> us Grid: <input type="text" value="1"/> us Freq: <input type="text" value="50.0"/> kHz Var: <input type="text" value="1.0"/> <input checked="" type="checkbox"/> Min d: <input type="text" value="14.0"/> us <input checked="" type="checkbox"/> Max d: <input type="text" value="28.0"/> us	
<input checked="" type="checkbox"/> Freq. stability eval on <input checked="" type="checkbox"/> Convergence check on Conv. margin: <input type="text" value="5"/> % Check span: <input type="text" value="1000"/>		<input checked="" type="checkbox"/> Min. distance eval on Min. distance: <input type="text" value="14"/> us <input checked="" type="checkbox"/> Convergence check on Conv. margin: <input type="text" value="5"/> % Check span: <input type="text" value="1000"/>		<input checked="" type="checkbox"/> Max distance eval on Max distance: <input type="text" value="28"/> us <input checked="" type="checkbox"/> Convergence check on Conv. margin: <input type="text" value="5"/> % Check span: <input type="text" value="1000"/>	
<input checked="" type="checkbox"/> PDF evaluation on Max freq. error: <input type="text" value="0"/> Max min. dist. error: <input type="text" value="0"/> Max max. dist. error: <input type="text" value="0"/>		<input type="checkbox"/> PDF evaluation (total) on <input checked="" type="checkbox"/> Convergence check on Conv. margin: <input type="text" value="5"/> % Check span: <input type="text" value="1000"/>		<input checked="" type="checkbox"/> Unique counter on Max freq. error: <input type="text" value="0"/> Max min. dist. error: <input type="text" value="0"/> Max max. dist. error: <input type="text" value="0"/>	
				<input type="checkbox"/> Unique counter (total) on <input type="button" value="Generate PateS scripts"/>	

>>> PateS GUI <<<
 (c) 2013 - 2015, Jacek Pierzchlewski
 Aalborg University, Denmark
<http://www.rfdacs.org/pates>

Figure 14: Graphical user interface to the Patterns Testing System (PATES). The system is available online in [21].

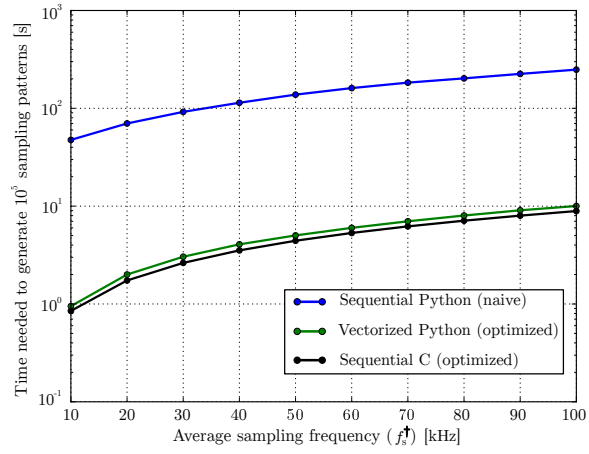


Figure 15: Time [seconds] needed to generate 10^5 sampling patterns vs. the average sampling frequency of sampling patterns.

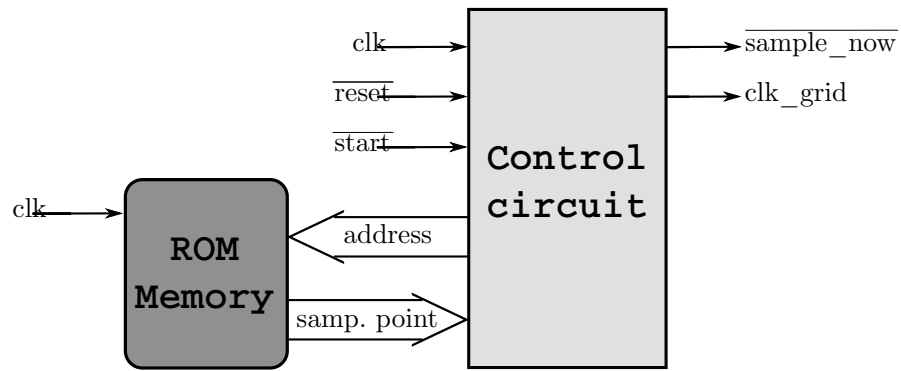


Figure 16: Block diagram of an implemented ADC driver.

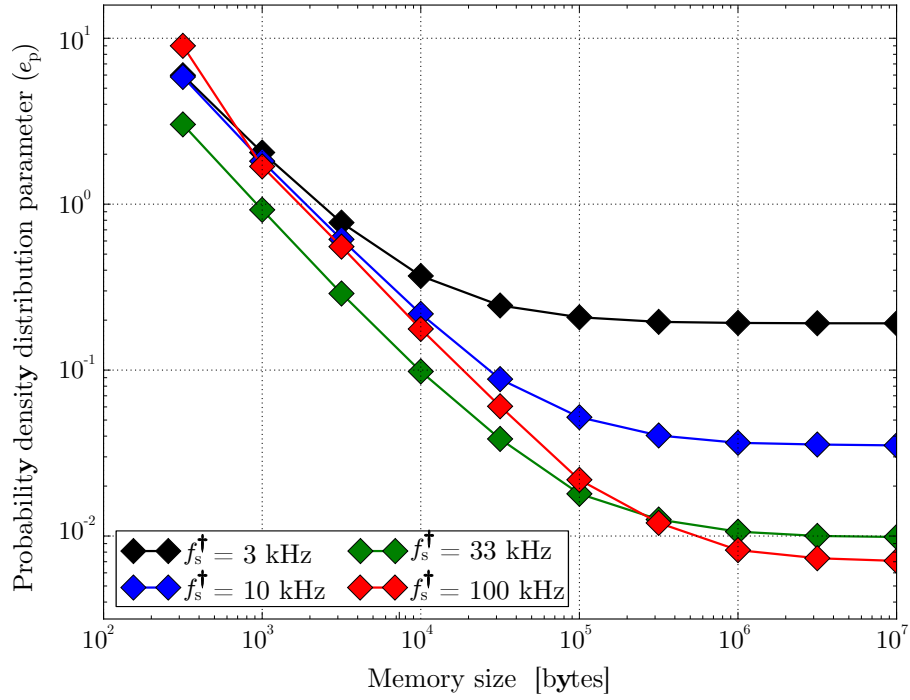


Figure 17: Probability density parameter e_p (19) found for patterns generated by the ANGIE algorithm vs. the size of memory for patterns storing.